Informe de Avance respecto al desafío dos.

Integrantes:

1. Juan José Mosquera.
2. Julián Andrés Sánchez Ballesteros.

Análisis del problema:

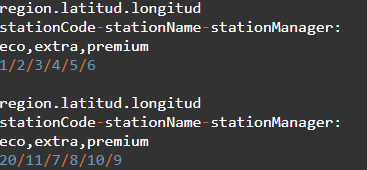
El primer paso fue leer la rúbrica, después de ser leída y comprender los ítems, como acto seguido se empezó el análisis de la estructura de nuestro programa teniendo en cuenta que se basa en POO, el resultado de dicho análisis fue la decisión de trabajar con 2 clases( estación y surtidor), nos inclinamos por dicha opción porque notamos que la gran mayoría de los requerimientos del programa son atributos de estas 2 clases anteriormente definidas, por lo tanto al desarrollar sus métodos podíamos abarcar una gran parte del problema, es importante aclarar que las 2 clases son independientes y no tienen una relación de herencia, y se pudo pensar que el surtidor puede ser una clase hija o derivada de estación, pero el motivo por el cual no se eligió este camino es porque son más los atributos propios del surtidor, es decir, atributos que no se derivan de la clase estación, entonces no tenía mucho sentido realizarlo de esta manera si no había muchos atributos de herencia entre las 2 clases. En los atributos de la clase surtidor se encuentran los detalles propios del surtidor como su modelo, código y estado, pero también se encuentran detalles respecto a las ventas, y la decisión de manejar clase estas últimas en esta es porque es la interacción directa entre el usuario y la adquisición de combustible, es decir, es el puente para que esto se pueda realizar, aunque esto también influye en la estación miremos que primero debe de pasar por el surtidor para llegar ahí, otro punto de vista es como ver que la venta se hizo en el surtidor, es decir, pertenece al surtidor; es por esto que decidimos hacerlo de esta manera.

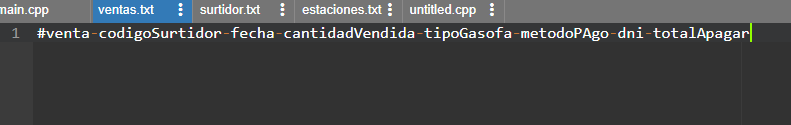
Almacenamiento de la información:

Para guardar los datos decidimos usar 3 archivos de texto plano(estaciones.txt, ventas.txt, surtidores.txt), ahora bien, ¿por qué usamos archivos?, porque a pesar de que sabemos la cuota superior de muchos valores como los surtidores o la capacidad del tanque, nosotros no sabemos cuántas estaciones se van a crear, o cuántas ventas se van a realizar, de hecho en la vida real lo que se calcula para saber el número de ventas es una probabilidad no un número exacto, entonces el uso de un arreglo para almacenar los datos nos puede traer 2 fallas, si la cantidad de posiciones en la memoria son muchas, y las que se van a ocupar son pocas se desperdicia memoria, o el caso opuesto, es decir, que haya más datos que posiciones de memoria, entonces no sería eficiente, más tarde vimos que para hacer esto era mejor utilizar vector, por el hecho de no preocuparnos por las posiciones de memoria, sin embargo, no estaba permitido, entonces apostamos por archivos de texto plano, en el cual había que preocuparse por el orden o la estructura que iban a tener los datos dentro del archivo y los métodos para extraer información de allí, así que aplicando lo aprendido en informática 1, separamos los datos por caracteres en específico como comas, puntos o espacios, esto para saber hasta dónde llega algún dato además otro beneficio es que, como ya saben un objeto muere después de la ejecución, por lo tanto si guardamos la información o el registro de esto en el archivo, no se perdería ningún dato,

la decisión de hacer 3 archivos es porque por ejemplo el registro de las ventas es un poco extenso, así que no era muy cómodo tener que manejarlo como una parte de otro archivo.

formato de estaciones.txt: lo que está separado por “/” son códigos de surtidores lo demás se evidencia en la imagen, cada información de estación se separa por una línea.



formato de ventas.txt:  


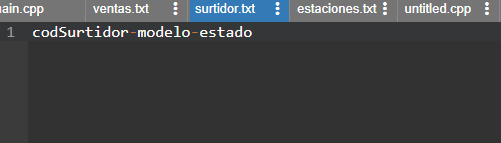
formato de surtidor.txt:  


Diagrama de clases:  
cómo las clases son independientes no hay ningún tipo de conexión entre ellas.

|  |
| --- |
| clase station |
| -string stationName.  -int stationCode.  -string stationManager.  - string stationRegion.  -float coordinatesGps[2].  -float stationTank[3]. |
| +station(string,int,string,string,float,float,float,float,float,int).  +~station().  +getters.  +setters. |

|  |
| --- |
| clase pump |
| -int code.  -int model.  - bool state.  -string date.  - float amountGas;  -int typeGas;  - int methodPay;  - int dni;  -float total; |
| +pump(int,int,bool,string,float,int,int,int,float);//Constructor  + ~pump();  +getters.  + setters. |